OPTIMAL COMPENSATOR DESIGN IN QUANTITATIVE FEEDBACK THEORY

(A report on Special Contract SPC-95-4032
submitted to the European Office of Aerospace Research and Development)

J N Ridley & A.L. Stevens

University of the Witwatersrand, Johannesburg

OPTIMAL COMPENSATOR DESIGN IN QUANTITATIVE FEEDBACK THEORY

(A report on Special Contract SPC-95-4032
submitted to the European Office of Aerospace Research and Development)

J N Ridley & A.L. Stevens

1. Introduction

The Quantitative Feedback Theory (QFT) technique developed by Isaac Horowitz over a number of years, is perhaps the only controller design methodology that enables a controller to be designed to a given specification in a transparent quantitative manner. By this is meant that there is a definite *quantitative* measure of the closeness of the design to an optimum. A major advantage of QFT is the fact that the trade-offs between the constraints and the set of design criteria are visible to the designer in a transparent manner at all stages *during* the actual design process, rather than at the end, as is the case with 'black box' *synthesis* techniques such as H_∞ or LQG optimal control.

The manual QFT method introduced by Horowitz and others in 1972 [3, 5] represented a major breakthrough in the quantitative design of robust controllers. However, the method is extremely labour intensive and the final loop-shaping stage of the design process requires substantial practice and expertise and it is believed that for this reason, the method has not been as widely accepted as it deserves to be.

This report details research carried out to develop a computer-based method for optimal loop-shaping in QFT. Although some work has already been done in this area by Gera and Horowitz [1] in 1980, no practical implementation details have been published. We believe that in OptComp we have made good progress in developing a program that enables the engineer to use QFT methods to design a compensator (or controller) iteratively to *any desired order,* while remaining transparent at all times about what trade-offs are necessary.

At each stage, the designer chooses the desired form of compensator, and the points in the Nichols chart through which it should pass. The program fits the best curve, and clearly displays the difference between the accuracy required and the accuracy attained, both graphically and in tabular form. The designer can then use his or her own judgement to decide whether a satisfactory result has been reached for a compensator of this order, and, if so, whether a compensator of higher order should be attempted. Thus, while all the drudgery has been removed from the design process, the designer still has the satisfaction of being closely involved in the art of design (for example, finding troughs in the boundaries) and in assessing trade-offs between accuracy at different ω-values, or between overall accuracy and the order of the controller. This means that the benefits of QFT are retained, and its drawbacks removed.

2. Objectives and contractual details

In terms of the original letter of 5 May 1996, the contract was in two parts:

a. To develop computer algorithms to enable computer-aided design of the optimum loop transmission for QFT design of multivariable control systems.

b. To incorporate the optimal loop transmission algorithms into the MIMOQFT design package developed at the US Air Force Institute of Technology.

In the formal contract the wording is somewhat different: to undertake preliminary development of algorithms for optimum computer-aided loop-shaping in control system design using QFT, with no mention of incorporation into MIMOQFT.

The starting point, as defined by Professor Houpis in a meeting at AFIT in October, is with a given nominal plant $P_0(j\omega)$, a given set of ω-values $\omega_1, \ldots, \omega_n$, and a set of composite boundaries $B_0(j\omega_1), \ldots, B_0(j\omega_n)$ [4, p. 12]. The novelty and originality of OptComp is is that it allows the user to design the compensator $G(j\omega)$ directly, rather than obtaining it indirectly as the quotient of $L_0(j\omega)$ and $P_0(j\omega)$ [4, p. 14].

The preliminary programs are written in Matlab, because of its power and simplicity (any deficiencies in precision are unimportant at the development stage), and because neither of us has convenient access to Mathematica at the moment. Once the programs have been finalized, translation into Mathematica and incorporation into the MIMOQFT package should be relatively minor tasks.

3. Mathematical description

It is assumed that the reader is familiar with at least the basic principles of QFT, as given, for example, in [2] and [4]. The conventional process of compensator design (or loop-shaping) is clearly described in [4, p. 14], and consists in finding the function $G(j\omega)$ so that

$$L_0(j\omega) = P_0(j\omega)G(j\omega), \quad \text{where} \quad G(j\omega) = \prod_{k=0}^{w} K_k G_k(j\omega)$$

[4, p.14], equation I.12. However, this process is indirect, in that the designer actually constructs $L_0(s)$, and "once a satisfactory $L_0(s)$ is achieved then the compensator is given by $G(s) = L_0(s)/P_0(s)$" [4, p.14].

We here employ an original approach of determining the compensator $G(s)$ directly, one factor $K_k G_k(s)$ at a time, until the desired order has been reached or the compensated function meets the required specifications. The method depends on two principles:

(a) The Nichols chart is essentially logarithmic (see Appendix 1), so $\text{nic}(L_0(j\omega)) = \text{nic}(P_0(j\omega)) + \text{nic}(G(j\omega))$. This means that the the requirement that $L_0(j\omega)$ lie on the boundary $B_0(j\omega)$ is equivalent to requiring $\text{nic}(G(j\omega))$ to lie on the shifted boundary $\text{nic}(B_0(j\omega)) - \text{nic}(P_0(j\omega))$ in the Nichols chart.

(b) $G(s)$ must be a rational function with pole excess greater than or equal to zero. By the Fundamental Theorem of Algebra, every polynomial with real coefficients can be factorized

as a product of linear and quadratic factors. It follows that $G(s)$ can be written as a product of rational functions of the six standard forms listed in the table below, and therefore

$$\mathrm{nic}(G(s)) = \sum_{k=0}^{w} \mathrm{nic}(K_k G_k(s)),$$

where each $K_k G_k(s)$ is of one of the six standard forms.

1.	constant/linear	$K/(s+b)$	(1st order lag)
2.	linear/linear	$K(s+a)/(s+b)$	(1st order lead-lag)
3.	linear/quadratic	$K(s+a)/(s^2+b_1 s+b_0)$	
4.	quadratic/quadratic	$K(s^2+a_1 s+a_0)/(s^2+b_1 s+b_0)$	(2nd order lead-lag)
5.	constant over quadratic	$K/(s^2+b_1 s+b_0)$	(2nd order lag)
6.		K/s	(pure integrator).

The six standard forms of compensator.

From the two observations (a) and (b) above comes the fundamental result, on which the program is based.

THEOREM. *Any compensator can be achieved by iteratively finding standard compensators* $K_k G_k(s)$ *for* $k = 1, 2, \ldots, w$ *such that* $\mathrm{nic}(K_k G_k(j\omega))$ *lies as close as possible to the shifted boundary* $\mathrm{nic}(B_0(j\omega)) - \mathrm{nic}(P_{k-1}(j\omega))$ *for all given values of* ω. *Here* $P_{k-1}(s)$, *for* $k > 1$, *denotes the partially compensated plant* $P_0(s) \prod_{r=1}^{k-1} K_r G_r(s)$.

In practice, the values $\omega_1, \ldots, \omega_n$ are given, as well as the plant values $P_0(j\omega_1), \ldots, P_0(j\omega_n)$, and the composite boundaries $B_0(j\omega_1), \ldots, B_0(j\omega_n)$. The first step is to find a standard compensator $K_1 G_1(s)$ so that $\mathrm{nic}(K_1 G_1(j\omega_r))$ lies as close as possible to the shifted boundary $\mathrm{nic}(B_0(j\omega_r)) - \mathrm{nic}(P_0(j\omega_r))$ for $r = 1, \ldots, n$. Qualitative knowledge of the shape of the Nyquist plots of the standard compensators in the Nichols chart is therefore essential, and representative samples are illustrated in Figure 3.1.

For minimum phase compensators, the plots all start with a phase angle of 0° when $\omega = 0$, and curve anticlockwise in the Nichols chart as ω increases. For the lead-lag compensators with zero pole excess (forms 2 and 4), the curves end with a phase angle of 0° as well, the difference being that form 2 curves have phase of one sign only, whereas form 4 curves can have both positive and negative phase, and cross the zero phase line at an intermediate value of ω. The curves for compensators with pole excess of one (forms 1 and 3) all approach the −90° phase line as $\omega \to \infty$. They differ only in that the phase decreases monotonically from 0° to −90° in form 1 curves, whereas in form 3 curves the phase first increases, then decreases beyond −90°, and increases again. Finally, form 5 curves approach the −180° phase line as $\omega \to \infty$, and form 6 plots lie on the −90° phase line, and do not require illustration.

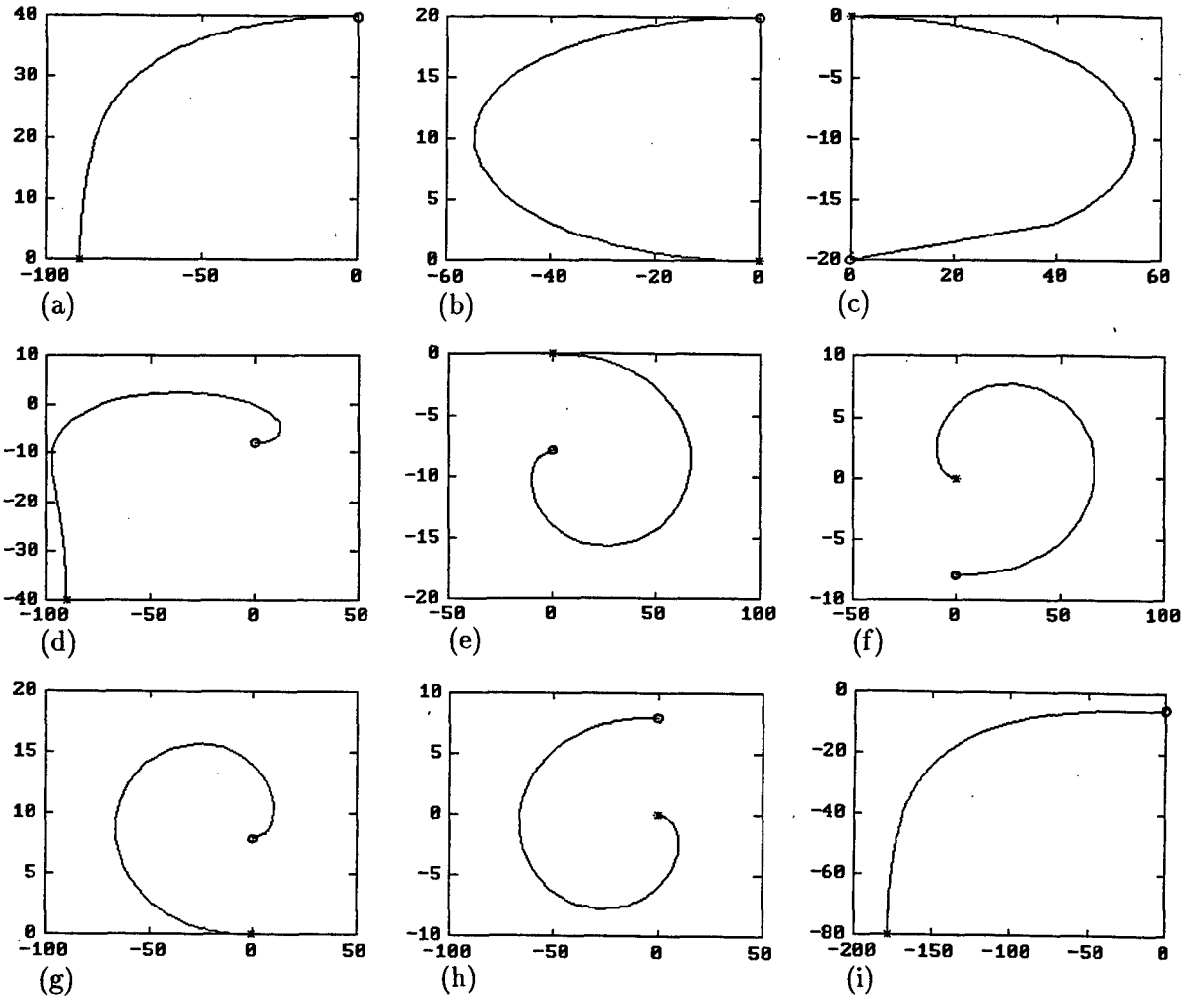


Figure 3.1. Nyquist plots of standard minimum phase compensators: (a) Form 1, (b)–(c) Form 2, (d) Form 3, (e)–(h) Form 4, (i) Form 5. The circle corresponds to $\omega = 0$ *and the star to* $\omega \to \infty$.

4. User's guide to OptComp

The program runs under Matlab, even under the student edition, if the boundary matrix is not too large. Once in Matlab, before the designer can use OptComp, the following data must be available:

- `omega` — a row vector of n values of ω, preferably in ascending order.
- `plant` — a row vector of n nominal plant transfer function values (in the Nichols chart) at $s = j\omega$, one for each value of ω stored in `omega`.
- `bndry` — an $m \times n$ matrix of composite boundary points in the Nichols chart, made up of m boundary points for each of the n values of ω. Note that, for convenience, each boundary has the same number of points; this can be achieved, if necessary, by interpolation, repetition, or omission.

Three sample data sets have been provided, and any one can be used (by entering `OptComp1`, `OptComp2`, or `OptComp3`) for immediate trial of the main program. All important output is

written into a diary file, which will be the default, named simply **diary**, unless another file name is specified with the **diary** command.

After entering `OptComp` at the Matlab prompt, the user is reminded of the facts in the preceding paragraphs, and given an option to exit. If he or she continues, the date and starting time are displayed, followed by a table showing the minimum compensation required (i.e., the magnitude and phase of the shortest distance from the nominal plant to the boundary) at each ω-value. The information is then repeated graphically in the Nichols chart in the conventional way, giving the Nyquist plot of the plant (in blue), with the points at the given ω-values marked by plus signs, and the boundaries drawn in white. The boundaries and plant points are each labelled with the corresponding ω-value. In order to keep the boundaries continuous, the Nichols chart is extended horizontally, if necessary, beyond the $\pm 180°$ phase angles. (The user can press any key to continue the program whenever it pauses for inspection of graphical or other display.)

As mentioned above, the originality of OptComp is that it enables the designer to concentrate directly on the compensation required at each ω-value. The following displays all concern the Nyquist plot (in the Nichols chart) of the compensator that is to be found. The first diagram shows the differences between the composite boundaries and the corresponding plant values. By the discussion in the previous section, the Nyquist plot of an optimal compensator will therefore lie on each of these shifted boundaries at the corresponding ω-value. In particular, if each of the shifted boundaries passes through the point $(0, 0)$, then no compensation is necessary, and an optimal compensator has been found.

After viewing this display, the designer is given the option to do nothing or to design a compensator of one of the six standard forms listed in the previous section. If the design requires a pure integrator in order to satisfy specifications on steady-state error, then the designer chooses option 6, and is prompted to enter the value of K in dB on the diagram by clicking the left mouse button at any point on the K dB line.

The option of using the first five standard forms is the heart of OptComp, and this is where the designer's skill and judgement are crucial. However, since the program runs quickly and gives clear graphical and tabular output, poor choices are immediately obvious and easy to discard. Thus experience and confidence are rapidly acquired.

After viewing the boundaries, the designer must decide on the lowest order standard compensator whose Nyquist plot seems most likely to pass close to them (in the correct order of frequencies). He or she is then prompted to use the mouse to enter on the diagram (in order) n points through which the Nyquist plot of the compensator should pass at the n values of ω. For optimal compensation, these points should be on the respective boundaries, and the judgement comes firstly in choosing the form of compensator, and secondly in selecting realizable points close to the boundaries. The program fits the best compensator of the desired form, and re-displays the previous diagram with the addition of the chosen points (green circles) and the Nyquist plot of the fitted compensator (red curve and stars). The green circles and red stars are all labelled with the appropriate ω-values. The designer can thus assess visually how good his or her choices were, and how effective the compensation is. (A warning message appears if the fitted compensator is not minimum phase.) More precise information is then given in

tabular form, displaying the minimum compensation required at each ω-value, firstly for the uncompensated plant and then with the new provisional compensator.

It is now easy for the designer to decide whether the new compensator is satisfactory. If not, then he or she can either quit, or repeat the step of choosing a form of compensator and selecting the points on the diagram through which its Nyquist plot should pass. To facilitate this choice at second and later attempts, the diagram also includes the red stars actually achieved by the previous unsatisfactory attempt. In this way, the designer is guided into a better choice the next time. With a little practice, one quickly becomes familiar with the standard Nyquist plots, and learns what choices are realizable in practice.

If the designer is satisfied with the compensation achieved by this form of compensator, then the gain and the coefficients of numerator and denominator are displayed and written to to the diary. The coefficients appear as vectors, so, for example, the form 4 compensator

$$\frac{123(s^2+4s+5)}{s^2+6s+10}$$

would appear as having gain `123`, numerator `1 4 5`, and denominator `1 6 10`. The previously displayed table of minimum errors, before and after this compensation step, is also written to the diary. Next, the Nyquist plot of the plant and composite boundaries, with which the process starts, is displayed again, now also including the Nyquist plot of the compensated plant (in green, with labelled circles at the ω-values). A warning message appears if the compensated plant is likely to have closed loop instability (i.e., if its Nyquist plot crosses the $\pm 180°$ phase line above the 0 dB line).

The designer can then quit (if this compensator has adequately met the design specifications) or repeat the process, starting with the already compensated plant, to increase the order of the compensator, i.e., to include an additional compensator of one of the standard forms.

5. Comments and conclusions

We believe that the procedures in OptComp can give the designer all the transparency of QFT and simplify the art of loop-shaping, while freeing him or her from laborious calculation. Experienced designers, familiar with traditional loop-shaping techniques, may need a period to adjust to the new approach of working directly with the compensator loop. It might be valuable to try the program with students, who do not have prejudices about other methods, to see how quickly they can acquire skill in loop-shaping using OptComp. Experimentation with the program will also have the advantage of suggesting where it can be improved, because many of its features have come from our own trials on sample data.

A full listing of `OptComp.m` appears in Appendix 3, and includes many comments to clarify the structure of the program and its various procedures. The curve-fitting step is certainly not the best possible, although the speedy trial-and-error method allows points giving a poor fit to be simply discarded and replaced by better choices. At this preliminary stage, the compensator coefficients are found by linearizing the equation in two ways, using Matlab's least squares procedure each time, then taking the average. Use of a sophisticated non-linear optimization routine might give closer fits.

References

1. Gera, Amos & Horowitz, Isaac, *Optimization of the loop transfer function*, Int. J. Control **31** (1980) 389–398.
2. Horowitz I. M., "Quantitative Feedback Design Theory", QFT Publications, 4470 Grinnell Avenue, Boulder, Colorado, USA, 1993.
3 Horowitz, I. & Sidi, M., *Synthesis of feedback systems with large plant ignorance for prescribed time domain tolerance*, Int. J. Control **16** (1972) 287–309.
4. Houpis, C.H. (ed.), "Quantitative Feedback Theory (QFT) for the Engineer" (WL-TR-95-3061), Flight Dynamics Directorate, Wright Laboratory, Wright-Patterson Air Force Base, OH, 1995.
5. Krishnan, K. & Cruikshanks, A., *Frequency domain design of feedback systems for specified insensitivity of time-domain response to parameter variation.* Int. J. Control, **25** (1977) 609–620.

University of the Witwatersrand, Johannesburg
Private Bag 3
WITS
2050 South Africa

Appendix 1. Glossary

compensator — A rational function of s, which has real coefficients, and in which the degree of the numerator is not greater than that of the denominator. A transfer function is multiplied by a compensator to make it meet predetermined specifications.

controller — Alternative term for compensator.

m-file — An executable Matlab file, with extension .m.

minimum phase — A minimum phase compensator is one with no poles or zeros in the right half-plane.

Nichols chart — Representation of complex numbers in terms of their phase angles (in degrees) and gain magnitudes (in dB, i.e., on a logarithmic scale). In the programs, each point in the Nichols chart is treated as a complex number phase +j gain. Except for the scales on the axes, the Nichols chart representation of a complex number is equal to $j\ln(\overline{z})$, so multiplication in the complex plane corresponds to addition in the Nichols chart. The m-files `nic.m` and `denic.m` respectively transform complex numbers to and from their Nichols chart representation, using the formulae
$\mathrm{nic}(z) = \frac{180}{\pi}\arg(z) + 20j\log_{10}|z|$ and
$\mathrm{denic}(w) = \exp(j\frac{\pi}{180}\mathrm{Re}(w) + 0.05\ln 10\,\mathrm{Im}(w))$.

Nyquist plot — The set of points obtained by transforming the imaginary axis by a given function. The Nyquist plot of f is the set $\{f(j\omega) \mid -\infty < \omega < \omega\}$. If $f(\overline{z}) = \overline{f(z)}$, then the plot is symmetrical, and only the portion for $\omega > 0$ is usually plotted. For our purposes, all Nyquist plots are of this form and are in the Nichols chart, i.e. $\{\mathrm{nic}(f(j\omega)) \mid \omega > 0\}$.

order — For a rational function, the sum of the degrees of numerator and denominator.

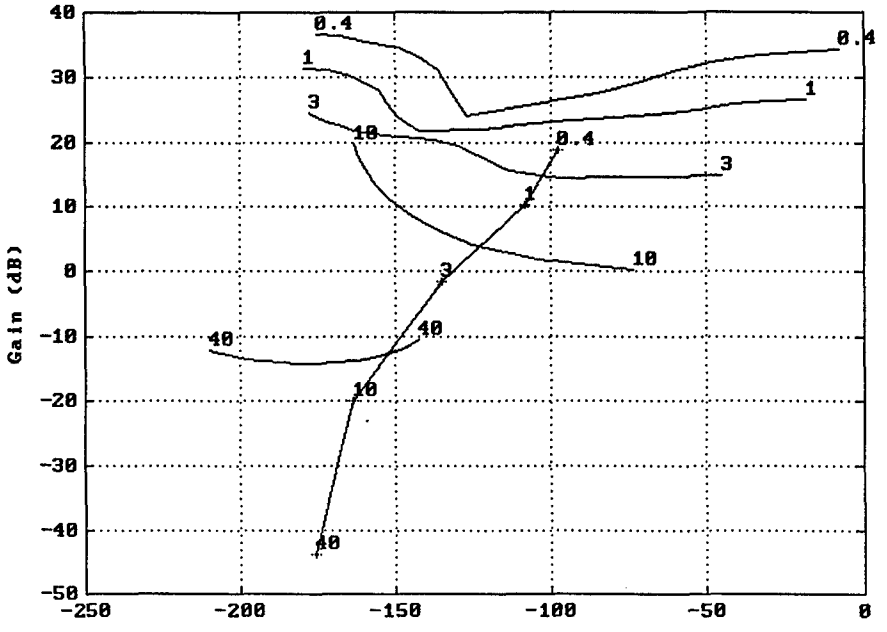
pole excess — For a rational function, the difference between the degrees of denominator and numerator.

QFT — Quantitative Feedback Theory.

Appendix 2. A typical OptComp session

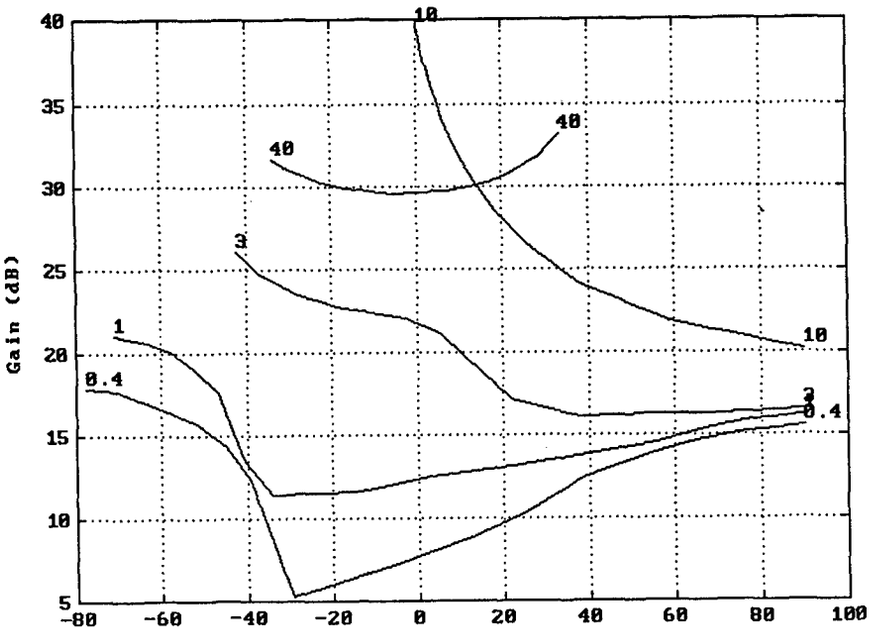
The following is the complete transcript of a session with OptComp, using the data loaded from Optcomp1. The total elapsed time is 19 minutes, most of which was spent waiting for the screen dumps of the graphical displays.

```
>> optcomp1
This plant can be compensated with a linear/linear compensator.
>> optcomp
Remember omega plant bndry must be defined beforehand.
Important output is recorded in a diary.

 Please enter 0 to quit or 1 to continue: 1
 12-Apr-96
Starting time:
    14    26

The next diagram gives the original plant values
and the composite boundaries at the values of omega indicated.
(Press any key.)
```




```
Minimum compensation required for optimal transfer function

     omega          dB      angle
    0.4000     5.2800   -29.3600
    1.0000    11.6200   -13.6800
    3.0000    16.0400    37.9500
   10.0000    20.1700    90.0000
   40.0000    29.5700    -5.2100
Optimal compensator values must lie on each of the curves shown in
```

the next diagram at the values of omega indicated. (Press any key.)



Please enter 0 to quit,

1 for constant/linear (1st order lag)

2 for linear/linear (1st order lead-lag)

3 for linear/quadratic

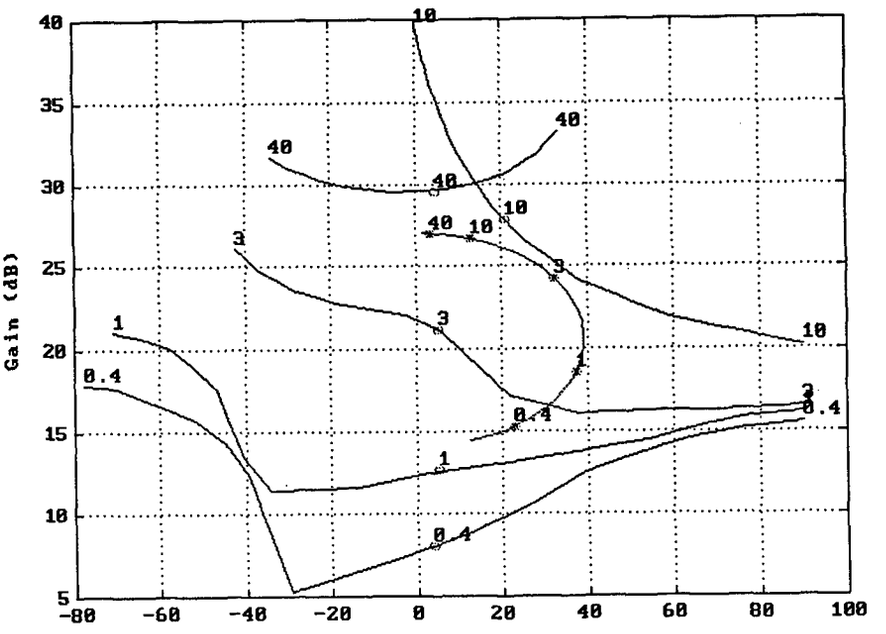
4 for quadratic/quadratic (2nd order lead-lag)

5 for constant over quadratic (2nd order lag)

6 for pure integrator (K/s)

2

Use the mouse to choose n points near the n boundaries shown. (Press any key.)

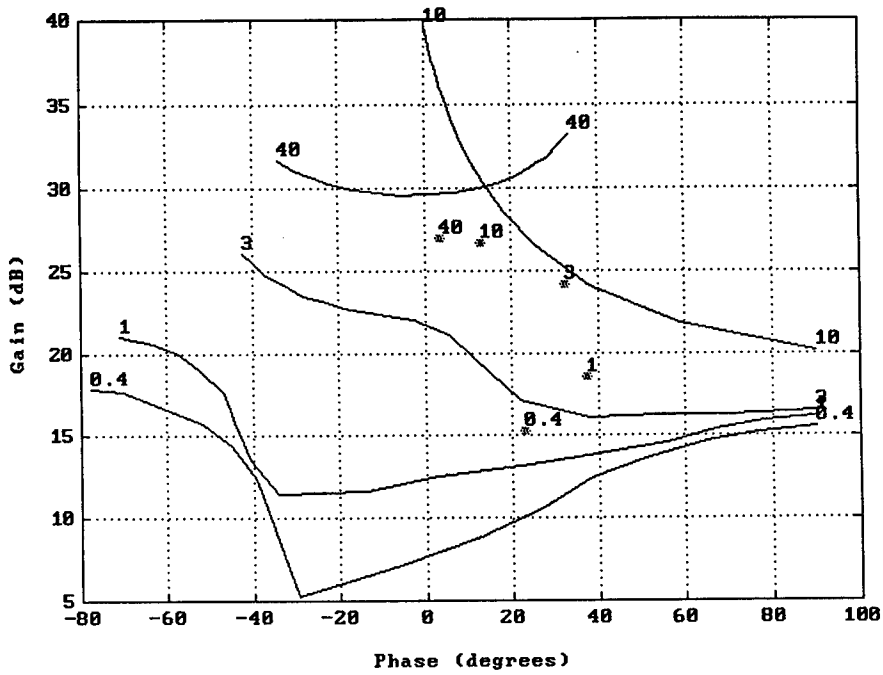


Minimum compensation required for optimal transfer function

	Previously		With this compensator	
omega	dB	angle	dB	angle
0.4000	5.2800	-29.3600	-2.8379	15.4838
1.0000	11.6200	-13.6800	-4.7049	4.4256
3.0000	16.0400	37.9500	-3.0035	-26.8701
10.0000	20.1700	90.0000	-0.1302	12.8050
40.0000	29.5700	-5.2100	2.6032	-3.2345

Please enter 0 if satisfied or 1 to replace this compensator or quit: 1

Optimal compensator values must lie on each of the curves shown in the next diagram at the values of omega indicated. (Press any key.)

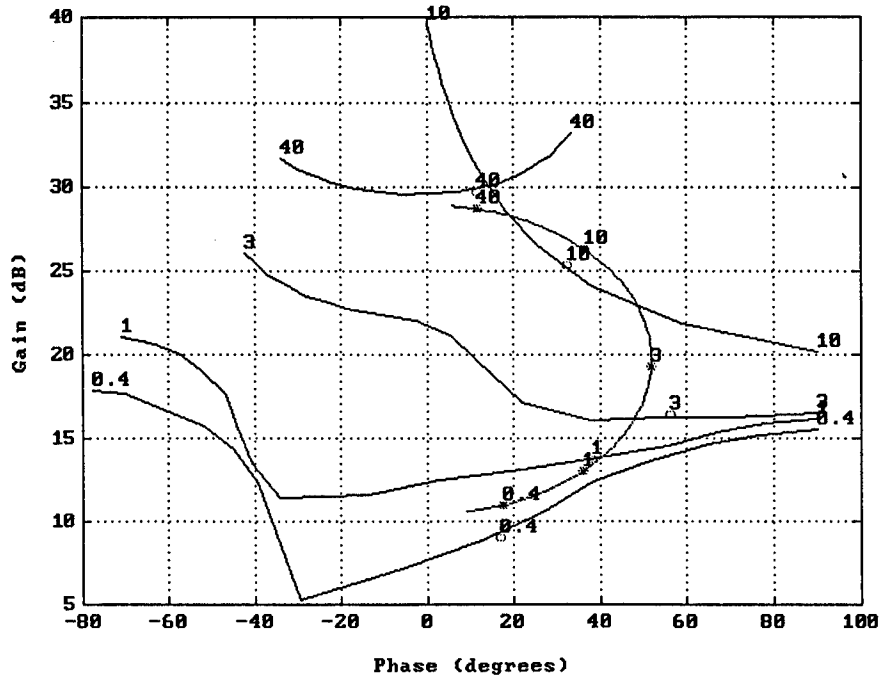


Please enter 0 to quit,
1 for constant/linear (1st order lag)
2 for linear/linear (1st order lead-lag)
3 for linear/quadratic
4 for quadratic/quadratic (2nd order lead-lag)
5 for constant over quadratic (2nd order lag)
6 for pure integrator (K/s)

2

Use the mouse to choose n points near the n boundaries shown.

(Press any key.)



Minimum compensation required for optimal transfer function

	Previously		With this compensator	
omega	dB	angle	dB	angle
0.4000	5.2800	-29.3600	-0.4857	9.0292
1.0000	11.6200	-13.6800	0.8641	5.5949
3.0000	16.0400	37.9500	-3.0216	2.5373
10.0000	20.1700	90.0000	0.1979	-10.5947
40.0000	29.5700	-5.2100	1.0117	-4.1553

Please enter 0 if satisfied or 1 to replace this compensator or quit: 0

Compensator gain

27.9524

Compensator numerator

1.0000 1.0976

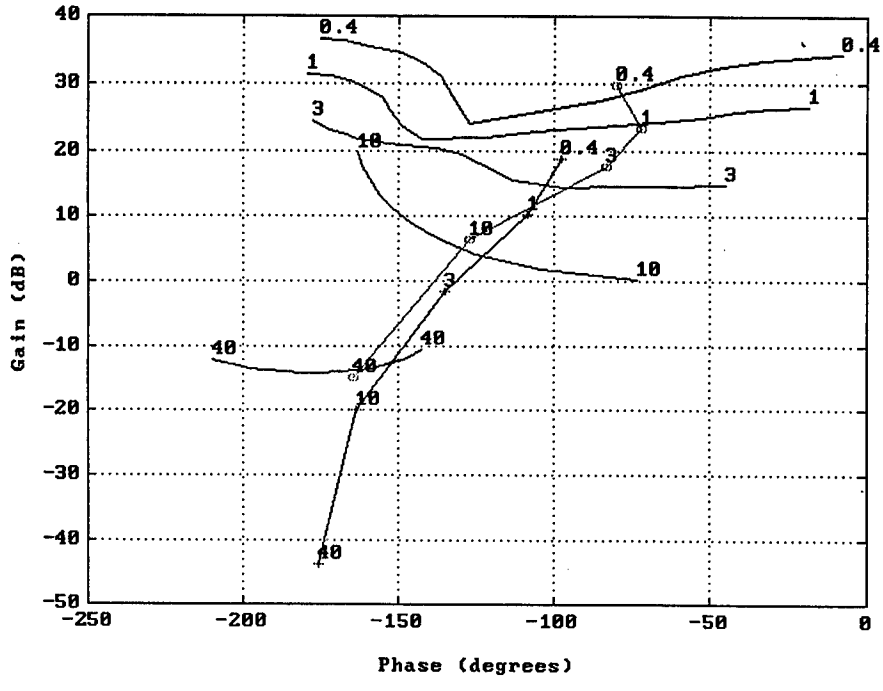
Compensator denominator

1.0000 9.1885

Minimum compensation required for optimal transfer function

	Previously		With this compensator	
omega	dB	angle	dB	angle
0.4000	5.2800	-29.3600	-0.4857	9.0292
1.0000	11.6200	-13.6800	0.8641	5.5949
3.0000	16.0400	37.9500	-3.0216	2.5373
10.0000	20.1700	90.0000	0.1979	-10.5947
40.0000	29.5700	-5.2100	1.0117	-4.1553

The next diagram gives the original and compensated plant values and the composite boundaries at the values of omega indicated. (Press any key.)



Please enter 0 to quit or 1 to design a higher order compensator: 0
Finishing time:
14 45

Appendix 3. Files on disk

The enclosed disk (in 1.44 MB $3\frac{1}{2}$ inch DOS format) contains six m-files, the main program (`OptComp.m`), two small function files (`nic.m` and `denic.m`) for conversion to and from the Nichols chart, and three sample data files (`OptComp1.m`, `OptComp2.m`, and `OptComp3.m`), any one of which can be run to provide input data for OptComp. A listing of the file `OptComp.m` follows.

```
%Program OPTCOMP.M
%This is an interactive program for designing a compensator (or controller)
%    for a given plant.
%Input data (to be defined beforehand) are:
%    omega (1 by n vector of angular velocities)
%    plant (1 by n vector of nominal plant values in Nichols chart,
%            one value for each value of omega)
%    bndry (m by n matrix of composite boundary points,
%            m boundary points for each value of omega)

%-----------------------PREAMBLE-----------------------------
disp('Remember omega plant bndry must be defined beforehand.')
disp('Important output is recorded in a diary.')
itrt = input('Please enter 0 to quit or 1 to continue: ');
if itrt < 1
  return
end
%--------------------INTRODUCTION----------------------------
diary on
disp(date)
temp = round(clock);
temp = temp([0 0 0 1 1 0]);
disp('Starting time:')
disp(temp)
diary off
[m,n] = size(bndry);
for l = 2:n
  if abs(real(plant(l)) - real(plant(l-1))) > 200.0 %%This extends the
    if real(plant(l)) > real(plant(l-1))             %Nichols chart, if
      plant(l) = plant(l) - 360.0;                   %necessary, to make
    else                                             %the plant locus
      plant(l) = plant(l) + 360.0;                   %continuous
    end                                              %
  end                                                %
end %l                                               %%
```

```
J = real(plant) > 180.0;                                %%This ensures that
temp = sum(J);                                           %the majority
if temp > 0.5*n                                          %of points lie between
  plant = plant - 360.0;                                 %the -180 and +180
else                                                     %degree phase lines
  J2 = real(plant) < -180.0;                             %
  temp2 = sum(J);                                        %
  if temp2 > 0.5*n                                       %
    plant = plant + 360.0;                               %
  end                                                    %
end                                                     %%
for l = 1:n
  for k = 2:m
    if abs(real(bndry(k,l)) - real(bndry(k-1,l))) > 200.0 %This extends
      if real(bndry(k,l)) > real(bndry(k-1,l))           %the Nichols chart,
        bndry(k,l) = bndry(k,l) - 360.0;                 %if necessary, to
      else                                               %make the
        bndry(k,l) = bndry(k,l) + 360.0;                 %boundaries
      end                                                %continuous
    end                                                  %
  end   % k                                              %
  J = real(bndry(:,l)) > 180.0;                         %%This ensures that
  temp = sum(J);                                         %the majority
  if temp > 0.5*m                                        %of points lie between
    bndry(:,l) = bndry(:,l) - 360.0;                     %the -180 and +180
  else                                                   %degree phase lines
    J2 = real(bndry(:,l)) < -180.0;                      %
    temp2 = sum(J);                                      %
    if temp2 > 0.5*m                                     %
      bndry(:,l) = bndry(:,l) + 360.0;                   %
    end                                                  %
  end                                                   %%
end %l                                                   %
disp('The next diagram gives the original plant values')
disp('and the composite boundaries at the values of omega indicated.')
disp('(Press any key.)')
pause
plot(real(bndry), imag(bndry),'-w',...
  real(plant),imag(plant),'-b',real(plant),imag(plant),'+b')
grid
xlabel('Phase (degrees)')
```

```
ylabel('Gain (dB)')
for l = 1:n
  text(real(bndry(1,l)),imag(bndry(1,l)),num2str(omega(l)))
  text(real(bndry(m,l)),imag(bndry(m,l)),num2str(omega(l)))
  text(real(plant(l)),imag(plant(l)),num2str(omega(l)))
end
pause
plant1 = plant;
ommin = 0.5*min(omega);
ommax = 2.0*max(omega);
omg = exp(linspace(log(ommin),log(ommax),100));

%------------------THE MAIN LOOP, DESIGNING ONE STANDARD COMPENSATOR
while itrt > eps  %this loop goes to the end of the program without indentation
plntmtrx = ones(m,1)*plant1;
diff = nic(denic(bndry - plntmtrx));                    %%The difference
for l = 1:n                                              %between boundaries
  for k = 2:m                                            %and plant values,
    if abs(real(diff(k,l)) - real(diff(k-1,l))) > 200.0%i.e. curves of optimal
      if real(diff(k,l)) > real(diff(k-1,l))             %compensation values
        diff(k,l) = diff(k,l) - 360.0;                   %
      else                                               %(Made continuous
        diff(k,l) = diff(k,l) + 360.0;                   % by extending beyond
      end                                                % +/-180 degrees
    end                                                  % if necessary.)
  end   % k                                             %%
  J = real(diff(:,l)) > 180.0;                          %%This ensures that
  temp = sum(J);                                         %the majority
  if temp > 0.5*m                                        %of points lie between
    diff(:,l) = diff(:,l) - 360.0;                       %the -180 and +180
  else                                                   %degree phase lines
    J2 = real(diff(:,l)) < -180.0;                       %
    temp2 = sum(J);                                      %
    if temp2 > 0.5*m                                     %
      diff(:,l) = diff(:,l) + 360.0;                     %
    end                                                  %
  end                                                   %%
end %l
rediff = real(diff);
imdiff = imag(diff);
temp2 = abs(denic(diff)-1.0);              %% This finds one
```

```
temp3 = ones(m,1)*min(temp2);                % point on each curve
J = abs(temp2-temp3)<eps ;                   % where the required
J2 = cumsum(J)<2;                            % compensation
J = J.*J2;                                   % has smallest magnitude.
error = nic(denic(diff(J)));                 %%
diary on
disp('Minimum compensation required for optimal transfer function')
disp('     omega         dB      angle')
disp([omega ; imag(error).' ; real(error).'].')
diary off
rpt = 2;
%------------------INNER LOOP, ALLOWING REPEATED ATTEMPTS----------
while rpt>eps
  disp('Optimal compensator values must lie on each of the curves shown in')
  disp('the next diagram at the values of omega indicated. (Press any key.)')
  pause
  plot(rediff, imdiff,'-w')
  grid
  xlabel('Phase (degrees)')
  ylabel('Gain (dB)')
  for l = 1:n
    text(rediff(1,l),imdiff(1,l),num2str(omega(l)))
    text(rediff(m,l),imdiff(m,l),num2str(omega(l)))
  end
  if rpt < 2
    hold
    plot(real(Gatt),imag(Gatt),'*r')
    for l = 1:n
      text(real(Gatt(l)),imag(Gatt(l)),num2str(omega(l)))
    end
    hold
  end
  pause
  disp('Please enter 0 to quit,')
  disp('             1 for constant/linear (1st order lag)')
  disp('             2 for linear/linear (1st order lead-lag)')
  disp('             3 for linear/quadratic')
  disp('             4 for quadratic/quadratic (2nd order lead-lag)')
  disp('             5 for constant over quadratic (2nd order lag)')
  disp('             6 for pure integrator (K/s)')
  nc = input(' ');
```

```
nc = round(nc);
if nc < eps
  rpt = 0;
else
  if nc == 6
    disp('The phase is constant at -90 degrees.')
    disp('Use the mouse to choose the numerator (i.e. the gain at omega = 1)')
    disp('(Press any key.)')
    pause
    [temp, K] = ginput(1);
    Gnum = [K];
    Gden = [1 0];
  else
    disp('Use the mouse to choose  n  points near the  n  boundaries shown.')
    disp('(Press any key.)')
    pause
    [rGreq,iGreq] = ginput(n);
    Greq = rGreq + i*iGreq;
    s = i*(omega');
    sinv = ones(n,1)./s;
    rhs = denic(Greq);
    if nc ==1
      %This fits a constant over linear compensator %%The curve-fitting
      Pcomp = [ones(n,1)  -rhs];                    %routines are fairly
      Qcomp = rhs.*s;                               %naive -
      P = [real(Pcomp);imag(Pcomp)];                %linearize the problem
      Q = [real(Qcomp);imag(Qcomp)];                %by cross-multiplying,
      cffs1 = P\Q;                                  %then break into real
      Pcomp = [sinv  -sinv.*rhs];                   %and imaginary parts,
      Qcomp = rhs;                                  %and use Matlab's
      P = [real(Pcomp);imag(Pcomp)];                %least squares fit.
      Q = [real(Qcomp);imag(Qcomp)];                %Then divide through
      cffs2 = P\Q;                                  %the highest power of s
      cffs = 0.5*(cffs1 + cffs2);                   %and repeat.
      Gnum = [cffs(1)];                             %Finally, average the
      Gden = [1 cffs(2)];                           %two answers.
    end  % of case nc = 1                          %%
    if nc == 2
      %This fits a linear over linear compensator
      Pcomp = [s ones(n,1) -rhs];
      Qcomp = rhs.*s;
```

```
  P = [real(Pcomp);imag(Pcomp)];
  Q = [real(Qcomp);imag(Qcomp)];
  cffs1 = P\Q;
  Pcomp = [ones(n,1) sinv -sinv.*rhs];
  Qcomp = rhs;
  P = [real(Pcomp);imag(Pcomp)];
  Q = [real(Qcomp);imag(Qcomp)];
  cffs2 = P\Q;
  cffs = 0.5*(cffs1 + cffs2);
  Gnum = [cffs(1) cffs(2)];
  Gden = [1 cffs(3)];
end  % of case nc = 2
if nc == 3
  %This fits a linear over quadratic compensator
  Pcomp = [s ones(n,1) -rhs.*s -rhs];
  Qcomp = rhs.*s.^2;
  P = [real(Pcomp);imag(Pcomp)];
  Q = [real(Qcomp);imag(Qcomp)];
  cffs1 = P\Q;
  Pcomp = [sinv sinv.^2  -sinv.*rhs -rhs.*sinv.^2];
  Qcomp = rhs;
  P = [real(Pcomp);imag(Pcomp)];
  Q = [real(Qcomp);imag(Qcomp)];
  cffs2 = P\Q;
  cffs = 0.5*(cffs1 + cffs2);
  Gnum = [cffs(1) cffs(2)];
  Gden = [1 cffs(3) cffs(4)];
end   % of case nc = 3
if nc == 4
  %This fits a quadratic over quadratic compensator
  Pcomp = [s.^2 s ones(n,1) -rhs.*s -rhs];
  Qcomp = rhs.*s.^2;
  P = [real(Pcomp);imag(Pcomp)];
  Q = [real(Qcomp);imag(Qcomp)];
  cffs1 = P\Q;
  Pcomp = [ones(n,1) sinv sinv.^2  -sinv.*rhs -rhs.*sinv.^2];
  Qcomp = rhs;
  P = [real(Pcomp);imag(Pcomp)];
  Q = [real(Qcomp);imag(Qcomp)];
  cffs2 = P\Q;
  cffs = 0.5*(cffs1 + cffs2);
```

```
    Gnum = [cffs(1) cffs(2) cffs(3)];
    Gden = [1 cffs(4) cffs(5)];
  end % of case nc == 4
  if nc == 5
    %This fits a constant over quadratic compensator
    Pcomp = [ones(n,1) -rhs.*s -rhs];
    Qcomp = rhs.*s.^2;
    P = [real(Pcomp);imag(Pcomp)];
    Q = [real(Qcomp);imag(Qcomp)];
    cffs1 = P\Q;
    Pcomp = [sinv.^2  -sinv.*rhs -rhs.*sinv.^2];
    Qcomp = rhs;
    P = [real(Pcomp);imag(Pcomp)];
    Q = [real(Qcomp);imag(Qcomp)];
    cffs2 = P\Q;
    cffs = 0.5*(cffs1 + cffs2);
    Gnum = [cffs(1)];
    Gden = [1 cffs(2) cffs(3)];
  end % of case nc == 5
end % of case nc > 0
if Gnum >= -eps
else
  disp('*****************WARNING*****************************')
  disp('This compensator has a zero in the right half-plane.')
  disp('*****************WARNING*****************************')
end
if Gden >= -eps
else
  disp('*****************WARNING*****************************')
  disp('This compensator has a pole in the right half-plane.')
  disp('*****************WARNING*****************************')
end
Gatt = nic(polyval(Gnum,i*omega)./polyval(Gden,i*omega));
Gcrv = nic(polyval(Gnum,i*omg)./polyval(Gden,i*omg));
if nc == 6
  Greq = Gatt;
end
plot(rediff, imdiff,'-w',real(Greq),imag(Greq),'og',real(Gatt),...
     imag(Gatt),'*r',real(Gcrv),imag(Gcrv),'-r')
grid
xlabel('Phase (degrees)')
```

```
    ylabel('Gain (dB)')
    for l = 1:n
      text(rediff(1,l),imdiff(1,l),num2str(omega(l)))
      text(rediff(m,l),imdiff(m,l),num2str(omega(l)))
      text(real(Greq(l)),imag(Greq(l)),num2str(omega(l)))
      text(real(Gatt(l)),imag(Gatt(l)),num2str(omega(l)))
    end
    pause
    newdiff = diff - ones(m,1)*Gatt;  %% This computes how much
    temp2 = abs(denic(newdiff)-1.0);   % extra compensation is
    temp3 = ones(m,1)*min(temp2);      % needed, if the current
    J = abs(temp2-temp3)<eps ;         % compensator is used.
    J2 = cumsum(J)<2;                  % (cf diff and error above)
    J = J.*J2;                         %
    newerr = nic(denic(newdiff(J)));  %%
    disp('Minimum compensation required for optimal transfer function')
    disp('                    Previously      With this compensator')
    disp('     omega          dB     angle         dB     angle')
    disp([omega' imag(error) real(error) imag(newerr) real(newerr)])
    rpt = input('Please enter 0 if satisfied or 1 to replace this compensator or
quit: ');
  end % of else  nc >= eps
end  % of while rpt > eps
%------------------END OF INNER LOOP-----------------------------

%------------------FINAL SECTION OF THE MAIN LOOP-------------------------------

if nc > eps                          %This section applies only if
  diary on                           %a standard compensator has
  if abs(Gnum(1)) > eps              %been designed.
    K = Gnum(1);                     %It gives the coefficients,
    Gnum = Gnum/K;                   %plots the boundaries and
    disp('Compensator gain');        %compensated and uncompensated
    disp(K)                          %plant loci,
  end                                %and tabulates the minimum
  disp('Compensator numerator')      %compensation necessary, before
  disp(Gnum)                         %and after introducing this
  disp('Compensator denominator')    %compensator.
  disp(Gden)
  disp('Minimum compensation required for optimal transfer function')
  disp('                    Previously      With this compensator')
```

```
disp('     omega        dB     angle        dB     angle')
disp([omega' imag(error) real(error) imag(newerr) real(newerr)])
diary off
plant1 = nic(denic(plant1 + Gatt));
temp = 0;
for l = 2:n
  if abs(real(plant1(l)) - real(plant1(l-1))) > 200.0
    if temp < 1
      if imag(plant1(l)) > 0
        disp('*****************WARNING*************************')
        disp('The closed loop transfer function may be unstable.')
        disp('*****************WARNING*************************')
        temp = 1;
      end
    end
    if real(plant1(l)) > real(plant1(l-1))
      plant1(l) = plant1(l) - 360.0;
    else
      plant1(l) = plant1(l) + 360.0;
    end
  end
end %l
J = real(plant1) > 180.0;                    %%This ensures that
temp = sum(J);                               %the majority
if temp > 0.5*n                              %of points lie between
  plant1 = plant1 - 360.0;                   %the -180 and +180
else                                         %degree phase lines
  J2 = real(plant1) < -180.0;                %
  temp2 = sum(J);                            %
  if temp2 > 0.5*n                           %
    plant1 = plant1 + 360.0;                 %
  end                                        %
end                                          %%
disp('The next diagram gives the original and compensated plant values')
disp('and the composite boundaries at the values of omega indicated.')
disp('(Press any key.)')
pause
plot(real(bndry), imag(bndry),'-w',...
  real(plant1),imag(plant1),'-g',real(plant1),imag(plant1),'og',...
  real(plant),imag(plant),'-b',real(plant),imag(plant),'+b')
grid
```

```
  xlabel('Phase (degrees)')
  ylabel('Gain (dB)')
  for l = 1:n
    text(real(bndry(1,l)),imag(bndry(1,l)),num2str(omega(l)))
    text(real(bndry(m,l)),imag(bndry(m,l)),num2str(omega(l)))
    text(real(plant1(l)),imag(plant1(l)),num2str(omega(l)))
    text(real(plant(l)),imag(plant(l)),num2str(omega(l)))
  end
  pause
end %of if nc > eps
%------------------END OF FINAL SECTION OF THE MAIN LOOP-----------

disp('Please enter 0 to quit or 1 to design a higher order compensator:')
itrt = input(' ');
end % of while itrt > eps
%------------------END OF THE MAIN LOOP----------------------------

diary on
temp = round(clock);
temp = temp([0 0 0 1 1 0]);
disp('Finishing time: ')
disp(temp)
diary off
```